分離論理入門のようなもの

そくらてす

2019 年 12 月 21 日
2020 年 8 月 22 日更新

1　分離論理とは

分離論理 (Separation Logic) とは述語論理をヒープ領域 (平たく言えば” メモリ”) に言及できるように拡張した論理体系である.

形式的には通常の述語論理 (一階述語論理など) にメモリの状態を記述できるような述語記号とそれらを結合するための分離結合子 $*$ などを付け加えたものと捉えられる.

プログラム検証の文脈で捉えると，分離論理は「ほど良い抽象性」を持ったヒープの状態の記法と言うことも出来る[*1].

この文書は一階述語論理の完全性定理くらいの内容を理解していて，かつ C 言語のポインタ概念を知っている人に対して，分離論理のさわりを解説するためのものである. 読者の参考になれば幸いである.

2　分離論理の言語とその意味論

この節では，分離論理の言語とその意味論を定義する.

今回の言語は分離論理の基礎概念の理解のために必要な部分だけを取り出したものである.

Definition 1 (分離論理の言語)
分離論理の言語を次のように定義する[*2].

Variables(変数記号)	$x \in \mathrm{Variables}$
Terms(項)	$t ::= \mathrm{nil} \mid x$
Atomic Formulæ(原子論理式)	$\alpha ::= \mathrm{emp} \mid t = t \mid t \overset{1}{\mapsto} \langle t \rangle \mid t \overset{2}{\mapsto} \langle t, t \rangle \mid \mathrm{ls}\,(t, t) \mid \mathrm{tree}\,(t)$
Formulæ(論理式)	$\varphi ::= \alpha \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi \mid \varphi * \varphi \mid \varphi \mathrel{-\!\!*} \varphi$

ヒープ領域を表す記号が増えている以外は一階述語論理とほぼ同様である. $\overset{i}{\mapsto}$ はポインタを表現する述語記号である. また，$\mathrm{ls}\,(\text{-}, \text{-})$, $\mathrm{tree}\,(\text{-})$ はリスト構造の存在と二分木構造の存在を表す述語記号である.

[*1] 応用上重要視されている Symbolic Heap と呼ばれる分離論理の部分体系は，通常使われるようなメモリの使い方をある程度カバーしつつ恒真性が決定可能になる体系である. 執筆時間の都合で Symbolic Heap についてこの文書ではこれ以上詳しく扱わない

[*2] この記法に慣れていない場合は「BNF 記法」でぐぐれ.

一階述語論理との最大の違いは分離結合子 (separating conjunction)$*$ と魔法の杖 (magic wand または separating implication) $-\!\!*$ である．分離結合子は異なるヒープ領域の状態の結合を表している．また，魔法の杖は前件部に書かれたヒープ領域の状態を結合すると後件が成り立つことを表している[*3]．

次に分離論理の意味論を定義する．

まず，一階述語論理の構造 (structure) と値割り当て (valuation) にあたるものを定義する．

Definition 2 (ヒープ領域モデル (Heap Memory Model))
次の条件を満たす 4 つ組 (Values, Location, s, h) を分離論理の構造と呼ぶ．

- Values，Location は空でない集合．
- $nil \in \text{Values} \setminus \text{Location}$.
- $s : \text{Variables} \to \text{Values}$.
- $h : \text{Location} \xrightarrow{\text{fin}} \text{Values} \cup \text{Values}^2$.

Values は変数記号の取り得る範囲を表す．Location はヒープ領域の番地 (Adress) の集合である．s は変数記号に対する値の割り当てである．h はヒープ領域を表す有限部分関数である．

本文書では Values や Location が具体的にどのような集合であるかは特に問題にはならない[*4]．ここ以降では

$$\begin{aligned} \text{Location} &:= \mathbb{Z}^+ \\ \text{Values} &:= \mathbb{Z} \cup \{nil\} \end{aligned}$$

と思っておけばよい (ただし，$\mathbb{Z}$ は整数全体の集合，$\mathbb{Z}^+$ は正の整数全体の集合)．

さて，分離論理の論理式の解釈を定義する．そのためにいくつかの略記を導入する．

$s : \text{Variables} \to \text{Values}$ に対して，

$$s\left[x \mapsto v\right](y) = \begin{cases} s(y) & \text{if } y \neq x \\ v & \text{if } y = x \end{cases}$$

と定める．

h_1，$h_2 : \text{Location} \xrightarrow{\text{fin}} \text{Values} \cup \text{Values}^2$ に対して $h_1 \perp h_2$ とは，$\text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset$ を意味する．

また，$h_1 \perp h_2$ に対して

$$h_1 * h_2(x) = \begin{cases} h_1(x) & \text{if } x \in \text{dom}(h_1) \\ h_2(x) & \text{if } x \in \text{dom}(h_2) \end{cases}$$

と定義する．

Definition 3 (分離論理の論理式の解釈)
構造 $\big(\text{Values}, \text{Location}, s, h\big)$，論理式 φ について，

$$s,\, h \models \varphi$$

という関係を次のように帰納的に定める．

[*3] ここで書かれた直感的な意味を読んで理解できなくても心配することはない．私自身，分離論理の勉強を始めたばかりの頃に，このように説明をされても正直よくわからなかった．分離論理の解釈の定義を読んだ後に戻って読むと意味がわかると思う．

[*4] むしろ，Values や Location の具体的な中身が問題となる場面をわたしは知らない．

以下 $[\![x]\!]_s = s(x)$，$[\![\mathsf{nil}]\!]_s = \mathrm{nil}$ とする．

$$
\begin{aligned}
s, h \models t_1 = t_2 &:\Longleftrightarrow [\![t_1]\!]_s = [\![t_2]\!]_s && (\text{if } \varphi \equiv t_1 = t_2)\\
s, h \models \neg\psi &:\Longleftrightarrow s, h \not\models \varphi && (\text{if } \varphi \equiv \neg\psi)\\
s, h \models \psi \wedge \vartheta &:\Longleftrightarrow s, h \models \psi \text{ and } s, h \models \vartheta && (\text{if } \varphi \equiv \psi \wedge \vartheta)\\
s, h \models \exists x.\psi &:\Longleftrightarrow \text{There exists } v \in \text{Values such that } s[\mathsf{x} \mapsto v], h \models \psi && (\text{if } \varphi \equiv \exists x.\psi)\\
s, h \models \mathrm{emp} &:\Longleftrightarrow \mathrm{dom}(h) = \emptyset && (\text{if } \varphi \equiv \mathrm{emp})\\
s, h \models t_0 \overset{1}{\mapsto} \langle t_1 \rangle &:\Longleftrightarrow \mathrm{dom}(h) = \{[\![t_0]\!]_s\},\ h([\![t_0]\!]_s) = [\![t_1]\!]_s && (\text{if } \varphi \equiv t_0 \overset{1}{\mapsto} \langle t_1 \rangle)\\
s, h \models t_0 \overset{2}{\mapsto} \langle t_1, t_2 \rangle &:\Longleftrightarrow \mathrm{dom}(h) = \{[\![t_0]\!]_s\},\ h([\![t_0]\!]_s) = ([\![t_1]\!]_s, [\![t_2]\!]_s) && (\text{if } \varphi \equiv t_0 \overset{2}{\mapsto} \langle t_1, t_2 \rangle)\\
s, h \models \psi * \vartheta &:\Longleftrightarrow \text{There exists } h_1 \perp h_2 \text{ where } h = h_1 * h_2,\\
&\qquad s, h_1 \models \psi \text{ and } s, h_2 \models \vartheta && (\text{if } \varphi \equiv \psi * \vartheta)\\
s, h \models \psi \mathrel{-\!\!*} \vartheta &:\Longleftrightarrow s, h' \models \psi \text{ implies } s, h * h' \models \vartheta \text{ for all } h' \perp h && (\text{if } \varphi \equiv \psi \mathrel{-\!\!*} \vartheta)\\
s, h \models \mathrm{ls}(t_1, t_2) &:\Longleftrightarrow s, h \models \mathrm{emp} \text{ or there esists } v \in \text{Values such that}\\
&\qquad s[x \mapsto v], h \models t_1 \overset{1}{\mapsto} \langle x \rangle * \mathrm{ls}(x, t_2) \text{ where } x \text{ is fresh} && (\text{if } \varphi \equiv \mathrm{ls}(t_1, t_2))\\
s, h \models \mathrm{tree}(t_1) &:\Longleftrightarrow s, h \models \mathrm{emp} \text{ or there esists } v_1,\ v_2 \in \text{Values such that}\\
&\qquad s[x \mapsto v_1, y \mapsto v_2], h \models t_1 \overset{2}{\mapsto} \langle x, y \rangle * \mathrm{tree}(x) * \mathrm{tree}(y)\\
&\qquad \text{where } x,\ y \text{ are fresh} && (\text{if } \varphi \equiv \mathrm{tree}(t_1))
\end{aligned}
$$

分離論理の論理式の解釈のうち，最初の 4 つは一階述語論理の論理式の解釈と同様である．

$s, h \models \mathrm{emp}$ の定義は emp がヒープ領域のすべてのノードが空であるということに対応していることを表している．

若干誤解が起きそうなのが，$s, h \models t_0 \overset{1}{\mapsto} \langle t_1 \rangle$ と $s, h \models t_0 \overset{2}{\mapsto} \langle t_1, t_2 \rangle$ の定義である．この定義では，$t_0 \overset{1}{\mapsto} \langle t_1 \rangle$ の意味は “Adress $[\![t_0]\!]_s$ にだけ値が入っており，その値が $[\![t_1]\!]_s$ である” と言っている．$s, h \models t_0 \overset{2}{\mapsto} \langle t_1, t_2 \rangle$ についても同様である．

一階述語論理における $P(\mathbf{c})$ の解釈の定義が “P の表現する集合に $\mathbf{c}$ が表しているモノが属している” というものであることに比べると，だけ という解釈の定義のおかげで分離論理のメモリに対する表現力は格段に強くなっている．では，$\mathrm{dom}(h) \supset \{[\![t_0]\!]_s\}$，$h([\![t_0]\!]_s) = [\![t_1]\!]_s$ という状況は表現できるのだろうか? 少し考えればわかるが $\vee, \neg, \exists$ などを組合せるだけではそのような状況は表現できない[*5]．その鍵は分離結合子 $*$ が握っている．

$s, h \models \psi * \vartheta$ の定義は，ヒープ領域が ψ で表現できる部分と ϑ で表現できる部分に “分離” できることを表している．これを使うと $\mathrm{dom}(h) \supset \{[\![t_0]\!]_s\}$，$h([\![t_0]\!]_s) = [\![t_1]\!]_s$ は $t_0 \overset{1}{\mapsto} \langle t_1 \rangle * \mathrm{True}$ で表現ができる (ただし，True は恒真な式)．

$s, h \models \psi \mathrel{-\!\!*} \vartheta$ の定義をみるだけでは $\mathrel{-\!\!*}$ の存在意義が分からないと思う．この $\mathrel{-\!\!*}$ の意義は entailment $\varphi * \psi \vdash \vartheta$ を考えることで分かる．

$\varphi \vdash \psi$ という記号列を entailment という (ただし φ，ψ は分離論理の論理式である)．任意の分離論理の構造に対して “$s, h \models \varphi$ ならば $s, h \models \psi$” が成り立つとき，entailment $\varphi \vdash \psi$ は真であるという．

さて，$\varphi * \psi \vdash \vartheta$ が真であるとき，$\varphi \vdash \psi \mathrel{-\!\!*} \vartheta$ は真であり，またその逆も成り立つ (要証明)．このことをふまえると $\mathrel{-\!\!*}$ は $*$ の “双対” と捉えることができる．

[*5] はず．少なくとも私は思い付かなかった．

最後に ls (-, -) と tree (-) はそれぞれヒープ領域におけるリスト構造の存在と二分木構造の存在を表す述語である．ls (-, -) は第一引数で始まり，第二引数で終わるようなリストが存在することを帰納的に定義している[*6]．同様に tree (-) は引数から始まる二分木構造の存在を示している．

3　分離論理の簡単な応用

この節では分離論理の簡単な応用として，簡単なプログラムの部分正当性を証明する．論理的にガチガチに固めて議論するとかえって分かりにくくなると思うので，ここから先はかなりラフに議論をしていく．

3.1　正しいプログラムとは

正しいプログラムとは何かということを考えよう．

プログラムを実際に書く時に必ず「仕様」というものを何らかの形で決める．**仕様**とは「そのプログラムにどのようなことをして欲しいか」を何かの形で決めたものである．プログラムが正しいというからには仕様を満たしていることが必要である[*7]．また, 何か変な入力をしたときにプログラムが永遠に走り続けて出力をしないというのも困る．

それをふまえて，次の条件を満たすものをこの文書では「正しいプログラム」と呼ぶことにする．

- プログラムが停止するとき，想定している計算結果を返す (部分正当性)
  - 別の言い方をすると仕様書の通りに動く[*8]
- いかなる入力に対しても停止する (停止性)
  - 入力によっては答えを返さずに永遠動き続けるということがない

書いたプログラムが正しいかどうかを確かめたいとき，普通はいくつかの想定される入力を実際に入力してみて想定される答えを返すかどうか，長い時間動きつづけないかどうかを試してみる．想定される入力の種類がそれほど多くなければ，すべてのパターンを試すこともできるだろうが，普通は人間の寿命の間ですべての入力を確かめられることはマレである．無限に思えるような膨大な入力に対しても “想定されている答え” を返してくれることを保証するにはどうすれば良いのか? 人類の持っている武器で適切なものが一つある．数学である．

数学は主に無限 (や有限ではあるが人類には無限に感じられるようなもの) を扱う学問なうえ，この世で最も「正しい」学問である[*9]．このように書かれたプログラムが「正しい」ことを数学的に保証しようとすることを**プログラム検証**と呼ぶ．

数学にプログラムが「正しい」ことを保証 (証明) してもらえれば世は事もなしである．しかも，数理論理学の知見では数学の証明は記号列の変形と見做せるので[*10]，その証明自体もプログラムで自動化できてしまえば万々歳で，人類はバグという二字をこの世から亡くすことができ，平和が訪れる．

[*6] もしかすると，人によってはこの定義に違和感を持つかもしれない．nil で終わるもののみを “リスト” とする定義に慣れている方である．これは，たとえば「x_1 から始まり，途中で x_2 が出現し，最後に nil で終わる」というようなリストを表現したいときがある．このとき，ここでの定義を採用しておくと $\mathrm{ls}\,(x_1, x_2) * \mathrm{ls}\,(x_2, \mathrm{nil})$ とそのリストを表現することができる．

[*7] プログラムの仕様がそのプログラムの要件を満たしているかどうかという難しい問題もあるが，ここでは考えない．

[*8] わかる人向け:もはや修正が不可能なほど書いてから気が付いてしまったのだが，メモリ安全性は部分正当性に入れてしまって良かったのだろうか.....?

[*9] この一文だけでいろんな人に刺されそう．

[*10] ホンマか?

だが，停止性については一般的に判定をするためのアルゴリズムがない，つまりプログラムによる自動化ができないことが知られている (停止性の決定不能性) [*11]．そのため，与えられたプログラムが「正しいプログラム」であるかどうかを判定することは一般的にはできない．

このことは，非常に残念なことである．が，停止性の決定不能性の意味するところが「どんなプログラムに対してもその停止性を保証できない」ということ**ではない**ということに注意すると，次善の策として「自動化出来る範囲で停止性を保証する」という方向性を見出すことができる．この試みはとても大事なものであるが，この文書ではこれ以上扱わない．

ここ以降では「停止性」のことは忘れて，一番目の「部分正当性」の保証について考える．分離論理が活躍するのはこちらサイドの問題を考えるときである．

「部分正当性」の問題を扱うにあたって重要な役割りを果たす道具のひとつが Hoare Triple である．

Definition 4 (Hoare Triple)
P, Q を (何らかの論理体系の) 論理式，C を (何かしらのプログラム言語で書かれた) プログラムとする．記号列

$$\{P\}\, C\, \{Q\}$$

を Hoare Triple と言い，P をこの Hoare Triple の事前条件 (precondition)，Q を事後条件 (postcondition) という．

気持ちとしては，「P という条件が成り立っているときにプログラム C を動かしたら Q が成り立つようになる」というのが Hoare Triple の意味するところである．この Hoare Triple が成立するかどうかを考える，つまり “Hoare Triple の証明を考える” ことでプログラムが部分正当性を検証するアプローチがある．

この Hoare Triple の事前条件と事後条件が分離論理の論理式で書かれたものを使ったプログラム検証の例を見せる．

3.2 死ぬほど単純なプログラム検証の例

本来であれば，検証するためのプログラムの定義を書かなければならない所だが，そこまでやるのはしんどいので C 言語風の疑似コードで書かれた関数についてプログラム検証（のようなもの）を行ってみせ雰囲気を伝えることにする．

次のような関数 `dells(*x)` について考える．

```
dells(*x){
  if (x = NULL) return 0;
  else{
    n := x.next;
    dells(n);
    free(x);
  }
}
```

[*11] この概念はすごく誤解を招くことでも有名である．正直わたしも正しく理解できているか不安である．

これは x から始まるリストを削除する関数である．このことを Hoare Triple を使って表すと次のようになる．

$$
\begin{array}{l}
\{\mathrm{ls}\,(x,\,\mathrm{nil})\} \\
\texttt{dells}(*\texttt{x})\{ \\
\quad \texttt{if}\,(\texttt{x} = \texttt{NULL})\ \texttt{return}\ 0; \\
\quad \texttt{else}\{ \\
\quad\quad \texttt{n} := \texttt{x.next}; \\
\quad\quad \texttt{dells}(\texttt{n}); \\
\quad\quad \texttt{free}(\texttt{x}); \\
\quad \} \\
\}\,\{\mathrm{emp}\}
\end{array}
$$

さて，本来であれば，この Hoare Triple に対して証明図を書くことでプログラムが「正しい」ことを証明するのであるが，手間を省くために各行にヒープ領域の状態を表す論理式を書くことでそれに代える．

最初のヒープ領域の状態が $\mathrm{ls}\,(x,\,\mathrm{nil})$ で表されるような状態であるとする．

プログラムの最初の行は $*\texttt{x}$ が NULL であるかどうかで条件分岐する if 文である．

$*\texttt{x}$ が NULL であるとき $\mathrm{ls}\,(x,\,\mathrm{nil})$ 定義からヒープ領域は $\{\mathrm{emp}\}$ という状態である．

そうでないならば，$\left\{\texttt{x} \overset{1}{\mapsto} \langle y\rangle * \mathrm{ls}\,(y,\,\mathrm{nil})\right\}$ という状態である．

$$
\begin{array}{l}
\{\mathrm{ls}\,(x,\,\mathrm{nil})\} \\
\texttt{dells}(*\texttt{x})\{ \\
\quad \texttt{if}\,(\texttt{x} = \texttt{NULL})\ \texttt{return}\ 0;\ \{\mathrm{emp}\} \\
\quad \texttt{else}\{\ \left\{\texttt{x} \overset{1}{\mapsto} \langle y\rangle * \mathrm{ls}\,(y,\,\mathrm{nil})\right\} \\
\quad\quad \texttt{n} := \texttt{x.next}; \\
\quad\quad \texttt{dells}(\texttt{n}); \\
\quad\quad \texttt{free}(\texttt{x}); \\
\quad \} \\
\}\,\{\mathrm{emp}\}
\end{array}
$$

$*\texttt{x}$ が NULL であるときヒープ領域の状態は emp になって return されるのでこれ以上変化しない．

$\texttt{x} \overset{1}{\mapsto} \langle y\rangle * \mathrm{ls}\,(y,\,\mathrm{nil})$ という状態のとき，次の文は x の指す先を n と置くというコマンドである．つまり，

$\mathtt{n} = y \wedge \mathtt{x} \overset{1}{\mapsto} \langle y \rangle * \mathrm{ls}\,(y,\, \mathrm{nil})$ という状態である．これは $\mathtt{x} \overset{1}{\mapsto} \langle \mathtt{n} \rangle * \mathrm{ls}\,(\mathtt{n},\, \mathrm{nil})$ と同値である．

$$
\begin{array}{l}
\{\mathrm{ls}\,(x,\, \mathrm{nil})\} \\
\mathtt{dells}(*\mathtt{x})\{ \\
\quad \mathtt{if}\,(\mathtt{x} = \mathtt{NULL})\ \mathtt{return}\ 0;\ \{\mathrm{emp}\} \\
\quad \mathtt{else}\{\ \left\{\mathtt{x} \overset{1}{\mapsto} \langle y \rangle * \mathrm{ls}\,(y,\, \mathrm{nil})\right\} \\
\qquad \mathtt{n} := \mathtt{x.next};\ \left\{\mathtt{x} \overset{1}{\mapsto} \langle \mathtt{n} \rangle * \mathrm{ls}\,(\mathtt{n},\, \mathrm{nil})\right\} \\
\qquad \mathtt{dells}(\mathtt{n}); \\
\qquad \mathtt{free}(\mathtt{x}); \\
\quad \} \\
\}\,\{\mathrm{emp}\}
\end{array}
$$

次の行は再帰呼び出し $\mathtt{dells}(\mathtt{n});$ である．$\mathtt{dells}(*\mathtt{x})$ の仕様から（正しく関数が書けていれば）$\mathrm{ls}\,(\mathtt{n},\, \mathrm{nil})$ を emp にするはずである．

$$
\begin{array}{l}
\{\mathrm{ls}\,(x,\, \mathrm{nil})\} \\
\mathtt{dells}(*\mathtt{x})\{ \\
\quad \mathtt{if}\,(\mathtt{x} = \mathtt{NULL})\ \mathtt{return}\ 0;\ \{\mathrm{emp}\} \\
\quad \mathtt{else}\{\ \left\{\mathtt{x} \overset{1}{\mapsto} \langle y \rangle * \mathrm{ls}\,(y,\, \mathrm{nil})\right\} \\
\qquad \mathtt{n} := \mathtt{x.next};\ \left\{\mathtt{x} \overset{1}{\mapsto} \langle \mathtt{n} \rangle * \mathrm{ls}\,(\mathtt{n},\, \mathrm{nil})\right\} \\
\qquad \mathtt{dells}(\mathtt{n});\ \left\{\mathtt{x} \overset{1}{\mapsto} \langle \mathtt{n} \rangle * \mathrm{emp}\right\} \\
\qquad \mathtt{free}(\mathtt{x}); \\
\quad \} \\
\}\,\{\mathrm{emp}\}
\end{array}
$$

もし，$\mathtt{dells}(*\mathtt{x})$ が正しく書けておらず $\mathrm{ls}\,(\mathtt{n},\, \mathrm{nil})$ を emp に出来ていないとするならばこのままヒープ領域の変化を追いかけていけば，異常なことが起こるはずである．

最後の $\mathtt{free}(\mathtt{x});$ によって $\mathtt{x} \overset{1}{\mapsto} \langle \mathtt{n} \rangle$ は emp に変わる．

$$
\begin{array}{l}
\{\mathrm{ls}\,(x,\, \mathrm{nil})\} \\
\mathtt{dells}(*\mathtt{x})\{ \\
\quad \mathtt{if}\,(\mathtt{x} = \mathtt{NULL})\ \mathtt{return}\ 0;\ \{\mathrm{emp}\} \\
\quad \mathtt{else}\{\ \left\{\mathtt{x} \overset{1}{\mapsto} \langle y \rangle * \mathrm{ls}\,(y,\, \mathrm{nil})\right\} \\
\qquad \mathtt{n} := \mathtt{x.next};\ \left\{\mathtt{x} \overset{1}{\mapsto} \langle \mathtt{n} \rangle * \mathrm{ls}\,(\mathtt{n},\, \mathrm{nil})\right\} \\
\qquad \mathtt{dells}(\mathtt{n});\ \left\{\mathtt{x} \overset{1}{\mapsto} \langle \mathtt{n} \rangle * \mathrm{emp}\right\} \\
\qquad \mathtt{free}(\mathtt{x});\ \{\mathrm{emp}\} \\
\quad \} \\
\}\,\{\mathrm{emp}\}
\end{array}
$$

以上のことから，if 文の分岐がどちらだったとしても最後には {emp} になることがわかったので $\mathtt{dells}(*\mathtt{x})$

が正しく書けていることがわかった.

$$
\begin{array}{l}
\{\mathrm{ls}\,(x,\,\mathrm{nil})\} \\
\texttt{dells}(*\texttt{x})\{ \\
\quad \texttt{if}\,(\texttt{x} = \texttt{NULL})\ \texttt{return}\ \texttt{0};\ \{\mathrm{emp}\} \\
\quad \texttt{else}\{\ \left\{\texttt{x} \overset{1}{\mapsto} \langle y\rangle * \mathrm{ls}\,(y,\,\mathrm{nil})\right\} \\
\qquad \texttt{n} := \texttt{x.next};\ \left\{\texttt{x} \overset{1}{\mapsto} \langle \texttt{n}\rangle * \mathrm{ls}\,(\texttt{n},\,\mathrm{nil})\right\} \\
\qquad \texttt{dells}(\texttt{n});\ \left\{\texttt{x} \overset{1}{\mapsto} \langle \texttt{n}\rangle * \mathrm{emp}\right\} \\
\qquad \texttt{free}(\texttt{x});\ \{\mathrm{emp}\} \\
\quad \}\ \{\mathrm{emp}\} \\
\}\,\{\mathrm{emp}\}
\end{array}
$$

注意

本来は前節のような証明は証明図を書くことによって行うが推論規則の定義などが面倒だったので割愛した。

また、ここまでの話だといちいち手動で証明をやっているように見えるが、実際のプログラム検証の目標はその証明を自動証明器などを用いて自動化することにある。

参考文献

[1] James Brotherston, “ An introduction to separation logic ”, Logic Summer School, ANU, 7 December 2015.

[2] John C. Reynolds, “Separation Logic: A Logic for Shared Mutable Data Structure”, Proceedings of the 17th Anual IEEE Symposium on Logic in Computer Science, 2002.